ISO 29990:2010

NATIONAL INFOCOMM COMPETENCY FRAMEWORK

NUS National University of Singapore

ISS INSTITUTE OF SYSTEMS SCIENCE

Advance Natural Language Processing

Introduction

09 Jan 2019

What is this course about?

- **Recent advance techniques for NLP**
  - esp. since the "*deep learning tsunami*"
- Getting a **good grasp of PyTorch for NLP**
  - Similar knowledge can be applied to any tensor libraries for machine/deep learning
- **Understanding the underlying techniques and knowing how to implement them**
  - Hopefully, you'll find them less "magical"

What is this course NOT about?

- **Understanding *everything* in NLP.**
  - There's a lot more to *computational linguistics* than what's in this course
- Using **libraries to achieve results for specific NLP tasks**
  - (e.g. SpaCy, AllenNLP, TorchText, etc.)
- No, **we won't build chatbots** in this course.

Pardon my quirks…

- **Sometimes I let the code do the talking and I just read out the code** (*Stop me if it's not intuitive enough*)

- **I have more code than math in my slides**

- **Feel free to stop me at any time to ask questions**

Some Questions…

- How many people learnt deep / machine learning from previous courses (in-class or MOOC)?

- How many of from computer science/engineering background? How many from humanities, STEM, non-computer engineering or business background? Others?

- How many currently working in a technology field? Industry/Academia?

Tentative Course Schedule

Introduction
- Classic vs Deep NLP
- NN from Scratch

Deep Learning Foundations
- Matrix Calculus for Deep Learning
- Backpropagation

Word Embeddings
- Word2Vec, GloVe, Fasttext, and friends
- Word Embeddings from Scratch

Nuts and Bolts
- Bias - Variance
- Regularization, Loss Functions, Optimizers

Recurrent Neural Nets
- RNN, LSTM, GRU
- Character and Subwords models

Attention Networks
- Attention
- Transformer

Language Models
- N-gram Language Models
- Elmo, BERT, UMLFiT and friends

Encoder-Decoder Models
- Machine Translation
- Teacher / Professor Forcing

Basysian Networks
- Bayesian Learning
- Variational Autoencoder

*Other Learning
- Reinforcement Learning
- Evolutionary Algorithms

Course Logistics

For the hands-on later, it'll take a while to download and install, so we do this first while the lesson goes on…

Go to https://www.anaconda.com/download

Download the Python3 version and install

Overview

Lecture

- Classic NLP (40 mins)
- Deep Magic NLP (20 mins)
- Deep Learning Basics (30 mins)

Hands-on

- Environment Setup (15 mins)
- Deep Learning From Scratch (60 mins)

Classic NLP

Vector space model, Frequency, TF-IDF and PPMI

Classic NLP

"***You shall know a word by the company it keeps…***"
– John R. Firth (1957)

Classic NLP

"***You shall know a word by the company it keeps…***"
– John R. Firth (1957)

"***Everyone is about Firth 1957 (you shall know a word…). Somehow we all skipped Firth 1935***"
– Yoav Goldberg (2016)

Classic NLP

"***You shall know a word by the company it keeps…***"
– John R. Firth (1957)

"***Everyone is about Firth 1957 (you shall know a word…). Somehow we all skipped Firth 1935***"
– Yoav Goldberg (2016)

"***… the complete meaning of a word is always contextual, and no study of meaning apart from context can be taken seriously***"
– John R. Firth (1935)

Classic NLP

"***The frequencies of word in a document tend to indicate the relevance of document to a query***"
– Gerard Salton (1975)

"***From frequency to meaning…. Statistical patterns of human word usage can be used to figure out what people mean***"
– Turney and Pantel (2010)

Vector Space Model

- Vector space models are **numerical representation of text**
- Traditionally, computed using **no. of times each word occurs**

Frequency (Count Vector)

```
sent0 = "The quick brown fox jumps over the lazy brown dog ."
sent1 = "Mr brown jumps over the lazy fox ."
```

	brown	dog	fox	jumps	lazy	mr	over	quick	the
sent0	2	1	1	1	1	0	1	1	2
sent1	1	0	1	1	1	1	1	0	1

Term Frequency – Inverse Document Frequency

$$tf_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if count(t,d)} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$idf_i = \log_{10} \frac{N}{df_i}$$

N: total no. of sentences

df_i: no. of sentences that contains word i

tf-idf value for word t in document d :

$$w_{t,d} = tf_{t,d} \times idf_t$$

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
from scipy.sparse.csr import csr_matrix

# The *TfidfVectorizer* from sklearn expects list of strings as input.
sent0 = "The quick brown fox jumps over the lazy brown dog .".lower()
sent1 = "Mr brown jumps over the lazy fox .".lower()

dataset = [sent0, sent1]

vectorizer = TfidfVectorizer(input=dataset, analyzer='word',
                ngram_range=(1,1), min_df = 0, stop_words=None)
tfidf_matrix =  vectorizer.fit_transform(dataset)

# Format the TF-IDF table into the pd.DataFrame format.
vocab = vectorizer.get_feature_names()
documents_tfidf_lol = [{word:tfidf_value
                        for word, tfidf_value in zip(vocab, sent)}
                       for sent in tfidf_matrix.toarray()]
documents_tfidf = pd.DataFrame(documents_tfidf_lol)
documents_tfidf.fillna(0, inplace=True)
```

Term Frequency – Inverse Document Frequency

```
sent0 = "The quick brown fox jumps over the lazy brown dog ."

sent1 = "Mr brown jumps over the lazy fox ."
```

	brown	dog	fox	jumps	lazy	mr	over	quick	the
sent0	0.500	0.351	0.250	0.250	0.250	0.000	0.250	0.351	0.500
sent1	0.354	0.000	0.354	0.354	0.354	0.497	0.354	0.000	0.354

Classification (Train/Test Split)

Train

```
sent0 = "The quick brown fox jumps over the lazy brown dog ."
sent1 = "Mr brown jumps over the lazy fox ."
sent2 = "Roses are red , the chocolates are brown ."
sent3 = "The frank dog jumps through the red roses ."
```

Test

```
sent4 = "Mr Tan jumps on red chocolates ?"
sent5 = "Mr brown likes the lazy dog ."
```

```
# Train sentences.
sent0, label0 = "The quick brown fox jumps over the lazy brown dog .".lower() , False
sent1, label1 = "Mr brown jumps over the lazy fox .".lower(), True
sent2, label2 = "Roses are red , the chocolates are brown .".lower(), False
sent3, label3 = "The frank dog jumps through the red roses .".lower(), False

# Test sentences.
sent4, label4 = "Mr Tan jumps on red chocolates ?".lower(), False
sent5, label5 = "Mr brown likes the lazy dog .".lower(), True

train_documents = [(sent0, label0), (sent1, label1),
                   (sent2, label2), (sent0, label2)]
test_documents = [(sent4, label4), (sent5, label5)]

train_texts, train_labels = zip(*train_documents)
test_texts, test_labels = zip(*test_documents)
```

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the TF-IDF features.
48  vectorizer = TfidfVectorizer(input=train_texts, analyzer='word',
49                               ngram_range=(1,1), # Note this parameter!
50                               min_df = 0, stop_words=None)
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Apply TF-IDF to the train and test set

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the TF-IDF features.
48  vectorizer = TfidfVectorizer(input=train_texts, analyzer='word',
49                               ngram_range=(1,1), # Note this parameter!
50                               min_df = 0, stop_words=None)
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Apply the Machine Learning algorithm

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the TF-IDF features.
48  vectorizer = TfidfVectorizer(input=train_texts, analyzer='word',
49                               ngram_range=(1,1), # Note this parameter!
50                               min_df = 0, stop_words=None)
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Accuracy = 50%

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the TF-IDF features.
48  vectorizer = TfidfVectorizer(input=train_texts, analyzer='word',
49                               ngram_range=(2,2), # Note this parameter!
50                               min_df = 0, stop_words=None)
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Lets use TF-IDF with **bigrams**

Term Frequency – Inverse Document Frequency

```
sent0 = "The quick brown fox jumps over the lazy brown dog ."

sent1 = "Mr brown jumps over the lazy fox ."
```

	brown dog	brown fox	brown jumps	fox jumps	jumps over	...	mr brown	the lazy	the quick
sent0	0.364	0.364	0.000	0.364	0.259	...	0.000	0.259	0.364
sent1	0.000	0.000	0.470	0.000	0.334	...	0.470	0.334	0.000

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the TF-IDF features.
48  vectorizer = TfidfVectorizer(input=train_texts, analyzer='word',
49                               ngram_range=(2,2), # Note this parameter!
50                               min_df = 0, stop_words=None)
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Accuracy = 100%

But we sort of "*cheated*" and we engineered the TD-IDF *features*

Pointwise Mutual Information

Does the w_1 and w_2 co-occur more than if they were independent?

$$\text{PMI}(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

Pointwise Mutual Information

Does the w_1 and w_2 co-occur more than if they were independent?

$$\mathrm{PMI}(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

$$P_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad P_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \quad PMI_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

Co-occurrence Matrix

```
sent1 = "Mr brown jumps over mr fox ."
```

	mr	brown	jumps	over	fox
mr					
brown					
jumps					
over					
fox					

Co-occurrence Matrix

```
sent1 = "Mr brown jumps over mr fox ."
```

	mr	brown	jumps	over	fox
mr		1	1		
brown					
jumps					
over					
fox					

Co-occurrence Matrix

```
sent1 = "Mr brown jumps over mr fox ."
```

	mr	brown	jumps	over	fox
mr		1	1		
brown			1	1	
jumps					
over					
fox					

Co-occurrence Matrix

```
sent1 = "Mr brown jumps over mr fox ."
```

	mr	brown	jumps	over	fox
mr		1	1		
brown			1	1	
jumps	1			1	
over					
fox					

Co-occurrence Matrix

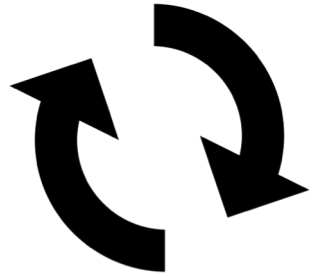
```
sent1 = "Mr brown jumps over mr fox ."
```

	mr	brown	jumps	over	fox
mr		1	1		
brown			1	1	
jumps	1			1	
over	1				1
fox					

Co-occurrence Matrix

```
sent2 = "Fox jumps over mr brown ."
```

	mr	brown	jumps	over	fox
mr		1	1		
brown			1	1	
jumps	1			1	
over	1				1
fox					



Repeat the same process for all sentences and add to the counts in the cells

Pointwise Mutual Information

		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
		mr	brown	jumps	over	fox
$i = 1$	mr	0	1	1	0	0
$i = 2$	brown	0	0	1	1	2
$i = 3$	jumps	1	0	0	2	1
$i = 4$	over	2	1	0	0	1
$i = 5$	fox	0	0	1	0	0

Matrix F with
- W rows (words)
- C columns (context)

Pointwise Mutual Information

		$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
		mr	brown	jumps	over	fox
$i=1$	mr	0	1	1	0	0
$i=2$	brown	0	0	1	1	2
$i=3$	jumps	1	0	0	2	1
$i=4$	over	2	1	0	0	1
$i=5$	fox	0	0	1	0	0

Matrix F with
- W rows (words)
- C columns (context)

$$\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij} = 15$$

Pointwise Mutual Information

		$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
		mr	brown	jumps	**over**	fox
$i=1$	mr	0	1	1	0	0
$i=2$	brown	0	0	1	1	2
$i=3$	**jumps**	1	0	0	**2**	1
$i=4$	over	2	1	0	0	1
$i=5$	fox	0	0	1	0	0

Matrix F with
- W rows (words)
- C columns (context)

$$P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij} = 15$$

P(jumps, over) = 2 / 15

Pointwise Mutual Information

		$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
		mr	brown	jumps	over	fox
$i=1$	mr	0	1	1	0	0
$i=2$	brown	0	0	1	1	2
$i=3$	**jumps**	**1**	**0**	**0**	**2**	**1**
$i=4$	over	2	1	0	0	1
$i=5$	fox	0	0	1	0	0

Matrix F with
- W rows (words)
- C columns (context)

$$P_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij} = 15$$

$$\sum_{j=1}^{C} f_{ij} = 4$$

$P(jumps, over) = 2 / 15$

P(jumps) = 4 / 15

Pointwise Mutual Information

		$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
		mr	brown	jumps	over	fox
$i=1$	mr	0	1	1	0	0
$i=2$	brown	0	0	1	1	2
$i=3$	jumps	1	0	0	2	1
$i=4$	over	2	1	0	0	1
$i=5$	fox	0	0	1	0	0

$$\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij} = 15 \qquad \sum_{j=1}^{C} f_{ij} = 4 \qquad \sum_{i=1}^{W} f_{ij} = 3$$

Matrix F with
- W rows (words)
- C columns (context)

$$P_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$P(jumps, over) = 2 / 15$

$P(jumps) = 4 / 15$

P(over) = 3 / 15

Pointwise Mutual Information

	mr	brown	jumps	over	fox
mr	0	1	1	0	0
brown	0	0	1	1	2
jumps	1	0	0	1.321	1
over	2	1	0	0	1
fox	0	0	1	0	0

$$P_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad P_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad PMI_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

P(jumps, over) = 2 / 15
P(jumps) = 4 / 15
P(over) = 3 / 15

***PMI(jumps, over)* = log (2/15) / (4/15 * 3/15) = 1.321**

Positive Pointwise Mutual Information

	mr	brown	jumps	over	fox
mr	0	1	1	0	0
brown	0	0	1	1	2
jumps	1	0	0	1.321	1
over	2	1	0	0	1
fox	0	0	1	0	0

P(jumps, over) = 2 / 15
P(jumps) = 4 / 15
P(over) = 3 / 15
PMI(jumps, over) = 1.321

$$P_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad P_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

$$P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad PMI_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

$$PPMI_{ij} = \begin{cases} PMI_{ij} & if \;\; PMI_{ij} > 0 \\ 0 & Otherwise \end{cases}$$

Pointwise Mutual Information

```
>>> import math

>>> math.log2(50/(100*200))
-8.643856189774725
```

$$PMI_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

Pointwise Mutual Information

```
>>> import math

>>> math.log2(50/(100*200))
-8.643856189774725

>>> math.log2(5/(10*20))
-5.321928094887363
```

$$PMI_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

(*higher is better*)

If we simply plug-in words with similar ratio

Pointwise Mutual Information

```
>>> import math

>>> math.log2(50/(100*200))
-8.643856189774725

>>> math.log2(5/(10*20))
-5.321928094887363

>>> math.log2(1/(2*4))
-3.0
```

$$PMI_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

(*higher is better*)

With the same ratio, PMI is biased towards rare words!!

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the TF-IDF features.
48  vectorizer = TfidfVectorizer(input=train_texts, analyzer='word',
49                               ngram_range=(2,2), # Note this parameter!
50                               min_df = 0, stop_words=None)
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the PPMI features.
48  vectorizer =PPMIVectorizer(input=train_texts, analyzer='word',
49                             context_window=2, stop_words=None)
50
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Replace the TF-IDF features with PPMI*

* There's no PPMIVectorizer in sklearn, you have to write it yourself

```
43  from sklearn.feature_extraction.text import TfidfVectorizer
44  import numpy as np
45  from scipy.sparse.csr import csr_matrix
46
47  # Extract the PPMI features.
48  vectorizer =PPMIVectorizer(input=train_texts, analyzer='word',
49                             context_window=2, stop_words=None)
50
51
52  X_train = vectorizer.fit_transform(train_texts)
53  X_test =  vectorizer.transform(test_texts)
54
55  y_train = train_labels
56  y_test = test_labels
57
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Machine learning block often don't change much

Classic NLP: Feature Engineering

- **Raw frequency is useful**
  - *but frequent words non-content words are not very informative*
- **TF-IDF resolves high freq non-content words issue**
  - *but each word is still somewhat independent of each other*
- **PPMI provides information about whether a word is informative in the context of another word**
  - *but biased towards infrequent events*

Classic NLP: Feature Engineering

- **TF-IDF and PPMI vectors are**
  - long (|V| > 100,000)
  - sparse (lots of zero)

- **Deep learning can create vectors that are**
  - short (often fixed-sized <2000, decided empirircally)
  - dense (most are non-zeros)

- **But it's not unlike *'modern'* deep learning based NLP**
  - one model improves upon another often incremental
  - they always come with certain caveats

Deep 'Magic' NLP

Recent advancement in Deep Learning in NLP

Classic NLP

"***The frequencies of word in a document tend to indicate the relevance of document to a query***"
– Gerard Salton (1975)

"***From frequency to meaning…. Statistical patterns of human word usage can be used to figure out what people mean***"
– Turney and Pantel (2010)

Deep 'Magic' NLP

"***The frequencies of word in a document tend to indicate the relevance of document to a query***"
– Gerard Salton (1975)

"***From frequency to meaning…. Statistical patterns of human word usage can be used to figure out what people mean***"
– Turney and Pantel (2010)

"We propose a unified NN architecture by trying to avoid task-specific engineering therefore disregarding a lot of prior knowledge"
– Collobert and Weston (2011)

Definitions (NLP, ML, DL)

- **Natural Language Processing (NLP) is …**
  - making computers understand and produce human languages.

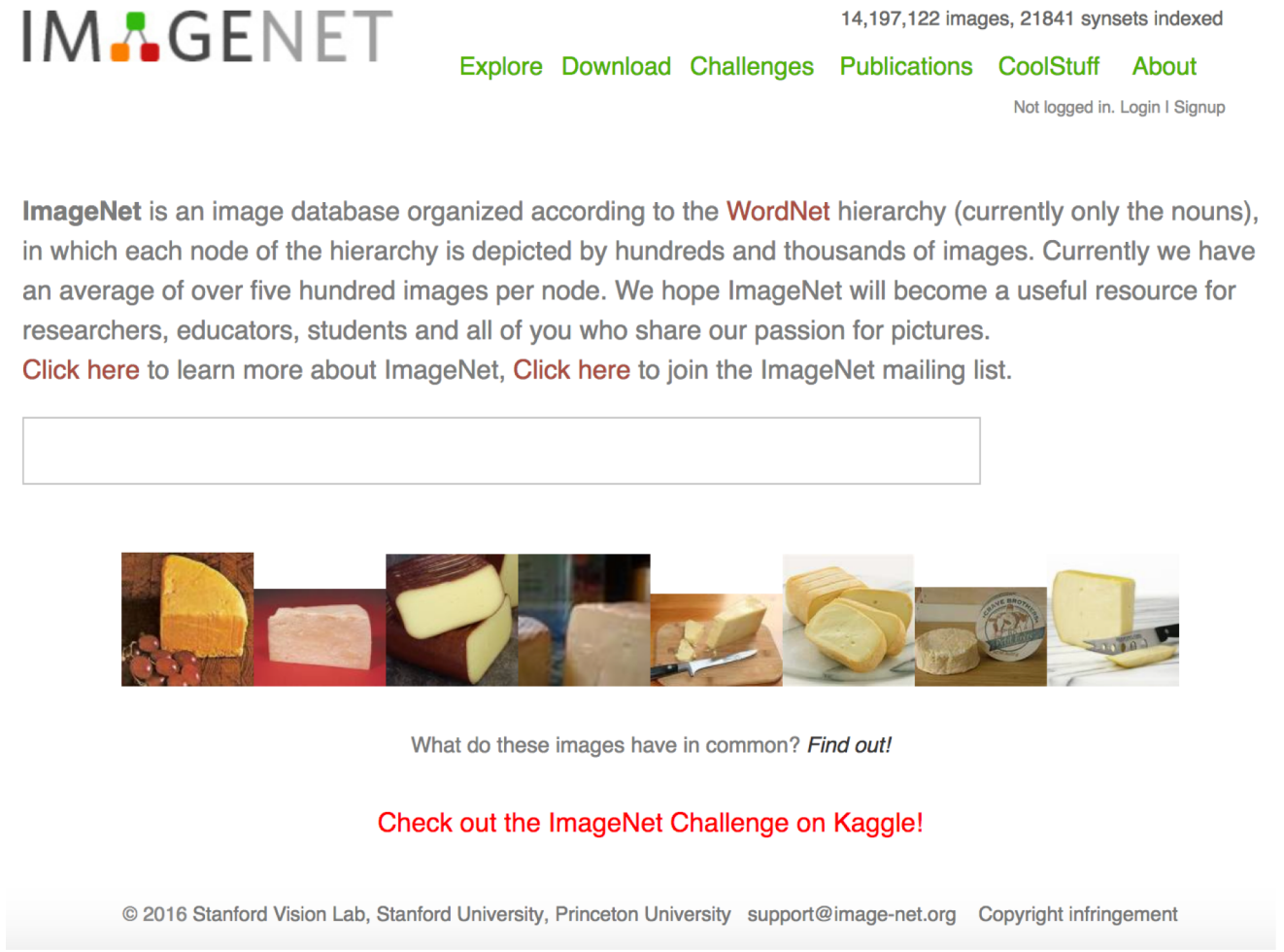
- **Machine Learning is …**
  - optimizing parameters/weights to best make a decision
  - *well-defined counting*

- **Deep Learning, some people say it's …**
  - neural nets
  - stacking multiple layers of "representation learning"
  - something that burns up as much GPUs as Bitcoin mining
  - a subset of methods in machine learning

The "ImageNet" Moment

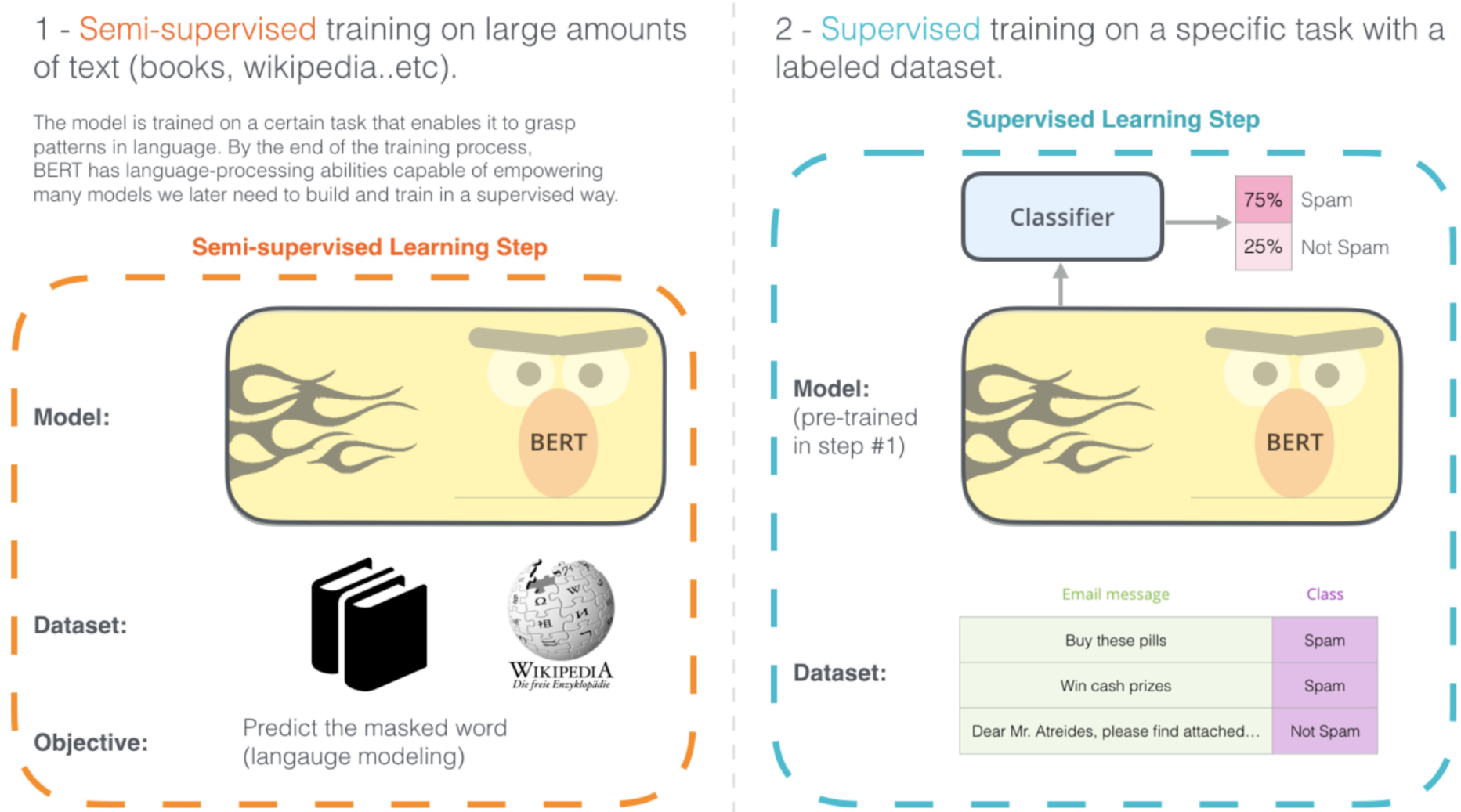
Various Computer Vision Challenges (ImageNet, MS Coco, etc.) started a wave of groups **training models and sharing pre-trained models.**

Fine-tuning / transfer learning for these pre-trained models are faster and *"usually better"* when training new models for other computer vision task.



(*Image from Stanford Vision Lab)

The "ImageNet" Moment for NLP



The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [Source for book icon].

(*Image from Jay Alammar's blog)

The "ImageNet" Moment for NLP

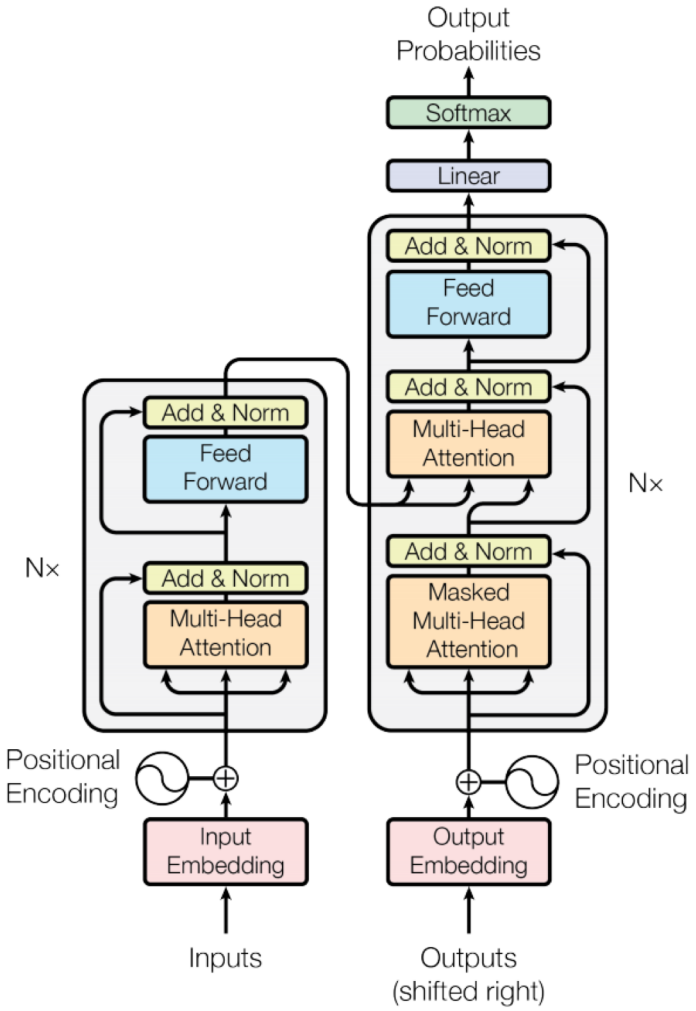


Figure 1: The Transformer - model architecture.

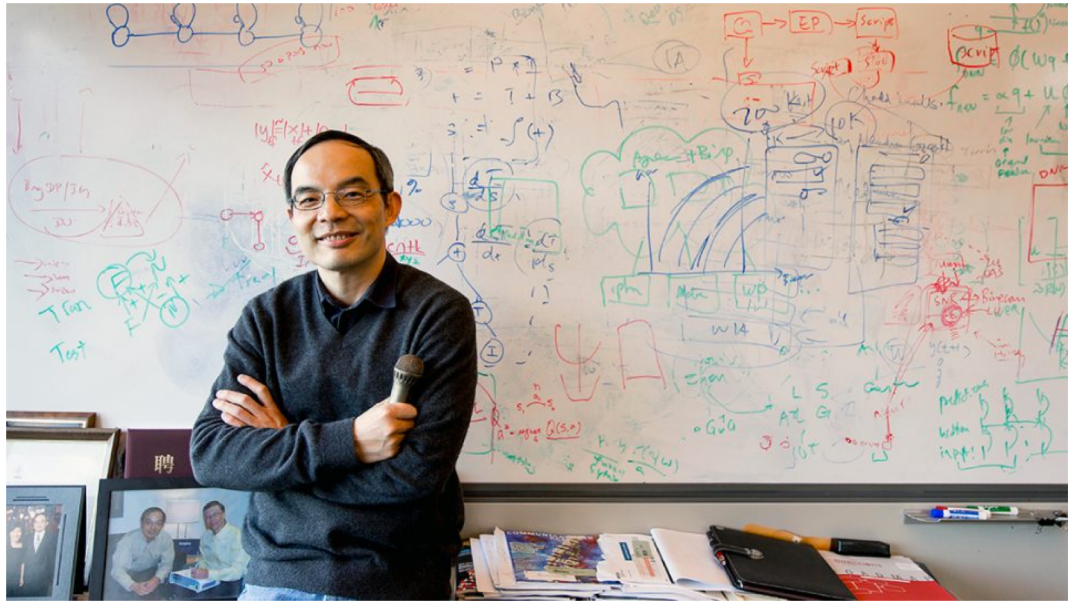
(*Image from Vaswani et al. 2017)

Major breakthrough of BERT came through the **self-attention network architecture, aka. Transformer** (Vaswani et al. 2017)

But do note the caveats with transfer-learning (aka. pre-training and fine-tuning):

- Could get **same results from random initialization** vs pre-training counterparts (He et al. 2018 on "*Rethinking ImageNet*")
- Understanding **why pre-training works in NLP still unclear** (Goldberg, 2018, see also Erhan, 2010)

Machine Translation Achieved Human Parity



(*Image from Microsoft AI Blog)

"Microsoft reaches a historic milestone, using AI to match human performance in translating news from Chinese to English"

- Microsoft AI Blog

Definition of Human Parity

If there is ***no statistically significant difference*** between human quality scores for … machine translation … and the scores for the corresponding human translations then the machine has achieved human parity.

Did MT really achieve Human Parity?

- **Has Machine Translation Achieved Human Parity? A Case for Document-level Evaluation**
  - Document level evaluation is necessary
  - http://aclweb.org/anthology/D18-1512

- **What Level of Quality can Neural Machine Translation Attain on Literary Text?**
  - Only 17-34% of novels can be machine translated
  - https://link.springer.com/chapter/10.1007/978-3-319-91241-7_12

- **Quality expectations of machine translation**
  - "*those of us who have seen many paradigms come and go know that overgilding the lily does none of us any good*"
  - https://arxiv.org/pdf/1803.08409.pdf

Did MT really achieve Human Parity?

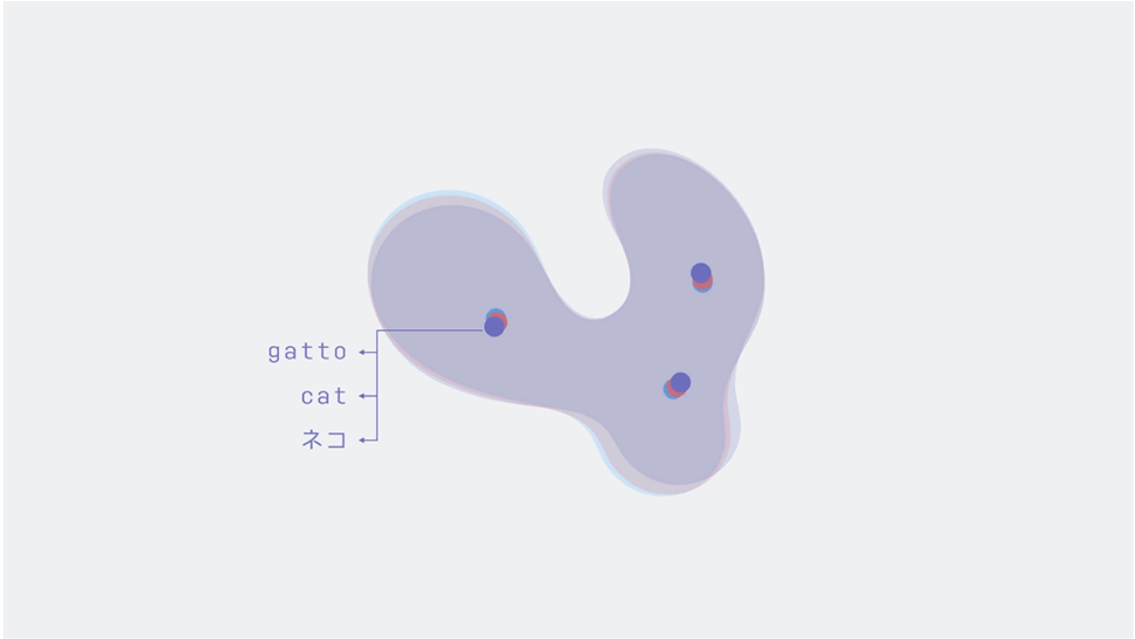
- **Quality expectations of machine translation**
  - *"those of us who have seen many paradigms come and go know that overgilding the lily does none of us any good"*
  - https://arxiv.org/pdf/1803.08409.pdf

- **"Human parity" in machine translation**
  - *"Some aspects of such translations are very good, but the frequent mistakes spoil things."*
  - http://languagelog.ldc.upenn.edu/nll/?p=40602

Machine Translation Achieved Human Parity

Algorithm 1: Unsupervised MT

1. **Language models:** Learn language models P_s and P_t over source and target languages;
2. **Initial translation models:** Leveraging P_s and P_t, learn two initial translation models, one in each direction: $P^{(0)}_{s \to t}$ and $P^{(0)}_{t \to s}$;
3. **for** k=1 **to** N **do**
4. **Back-translation:** Generate source and target sentences using the current translation models, $P^{(k-1)}_{t \to s}$ and $P^{(k-1)}_{s \to t}$, factoring in language models, P_s and P_t;
5. Train new translation models $P^{(k)}_{s \to t}$ and $P^{(k)}_{t \to s}$ using the generated sentences and leveraging P_s and P_t;
6. **end**



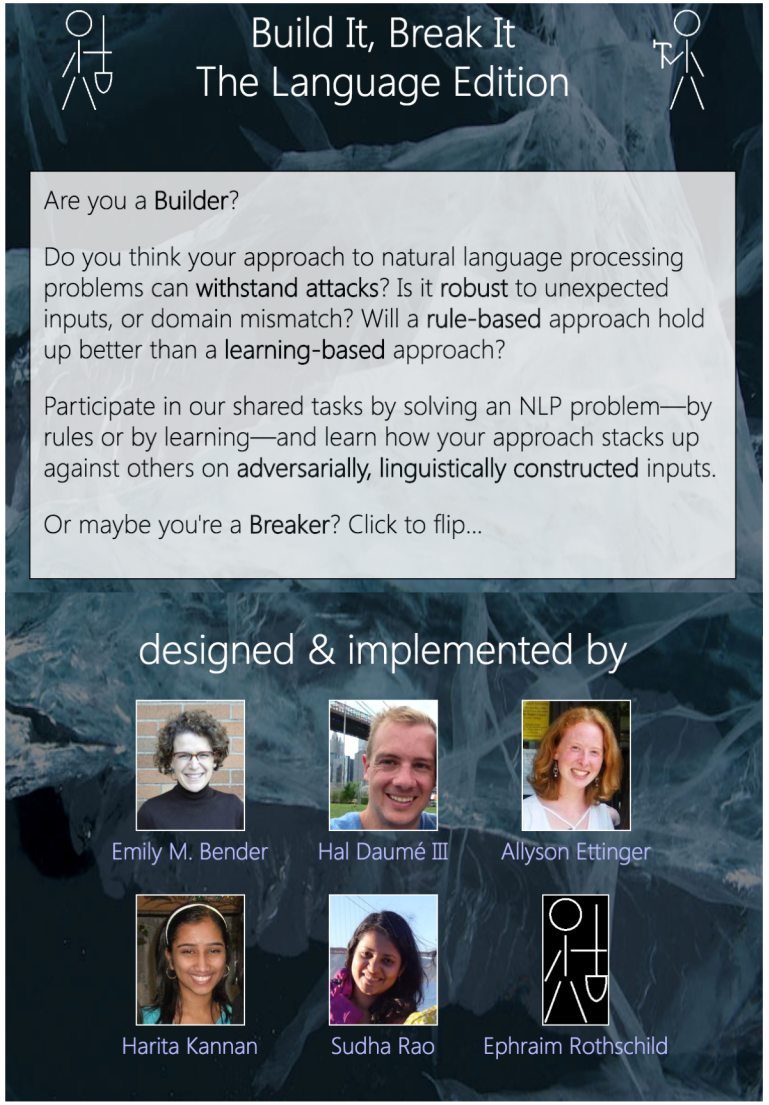
(*Image from Facebook Code Blog)

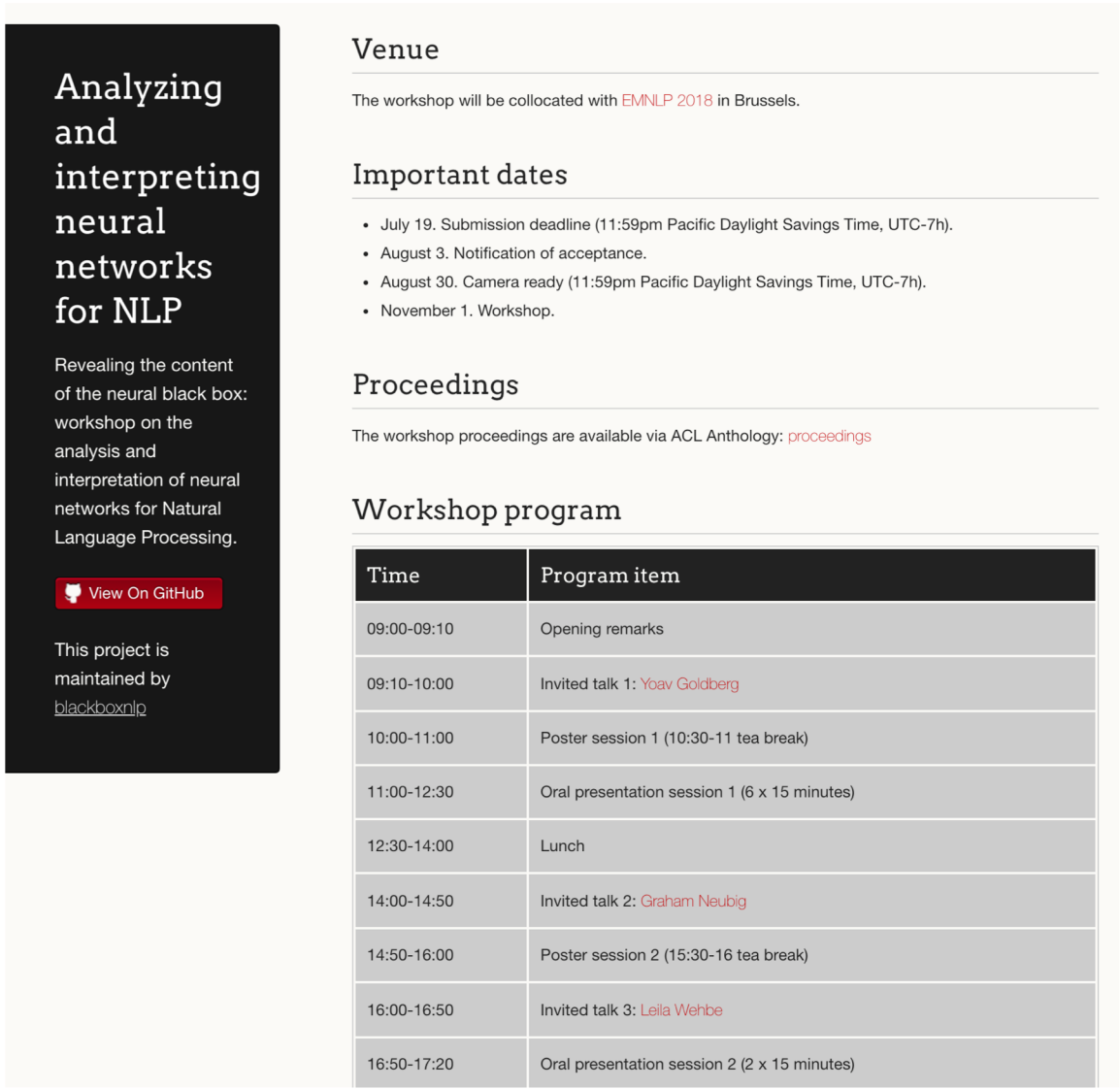
"Two-dimensional word embeddings in two languages can be aligned via a simple rotation. After the rotation, word translation is performed via nearest neighbor search."
– Facebook Code Blog

Unsupervised Machine Translation

- **Unsupervised SMT (IXA NLP Group)**
  - Non-neural: http://aclweb.org/anthology/D18-1399
- **Unsupervised NMT (IXA NLP and Cho)**
  - https://arxiv.org/pdf/1710.11041.pdf
- **Unsupervised NMT with weights sharing (UCAS)**
  - http://www.aclweb.org/anthology/P18-1005
- **Lots of unsupervised MT tech at Facebook at EMNLP 2018**
  - https://research.fb.com/facebook-research-at-emnlp/

Robustness and Interpretability





Deep Learning Basics

Recent advancement in Deep Learning in NLP

Perceptron

Perceptron algorithm is a:

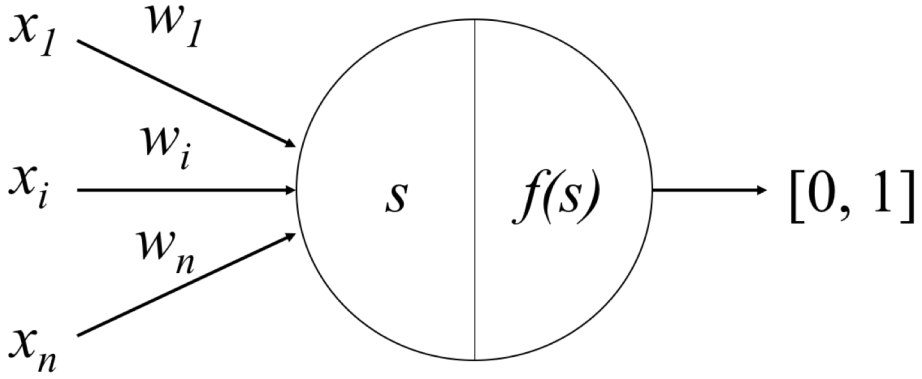
> "*system that depends on **probabilistic** rather than deterministic principles for its operation, gains its reliability from the **properties of statistical measurements obtain from a large population of elements***"
> - Frank Rosenblatt (1957)

```
58  # Pick your poison.
59  from sklearn.linear_model import Perceptron
60  # Initialize your classifier.
61  clf = Perceptron(max_iter=10)
62  # Train the classifier.
63  clf.fit(X_train, y_train)
64
65  print(clf.predict(X_test))
```

Perceptron

Given a **set of inputs x**, perceptron

- learns ***w* vector** to map the inputs to a real-value output between $[0,1]$
- through the **summation of the dot product of the *w*·*x***
- with a **transformation function** (aka. activation function)



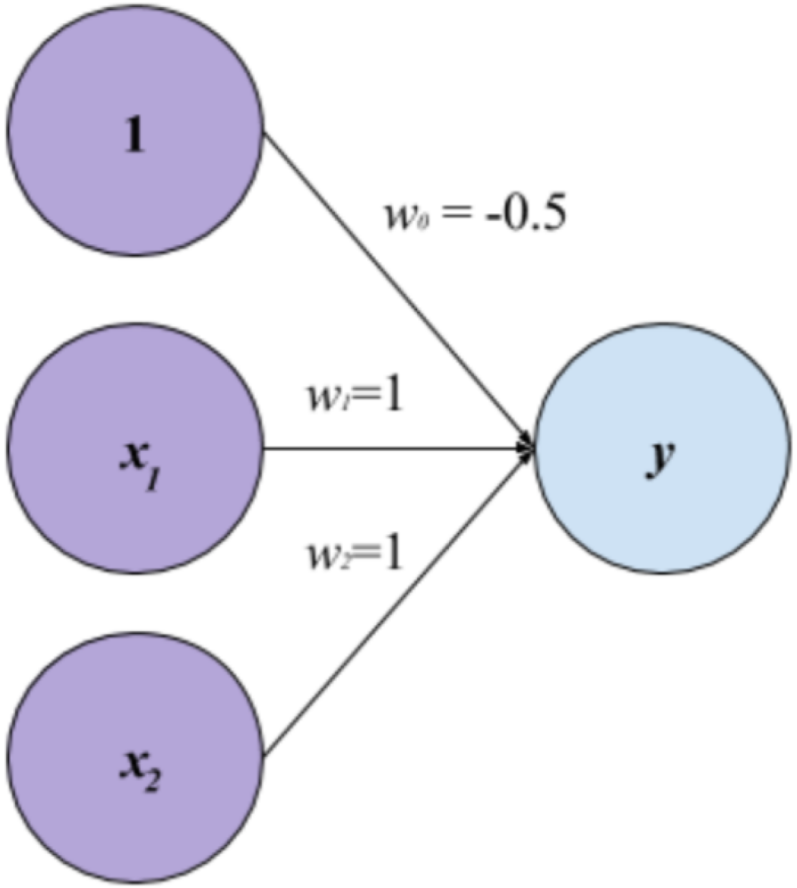
Summation

$$s = \sum w \cdot x$$

Transformation

$$f(s) = \frac{1}{1+e^{-s}}$$

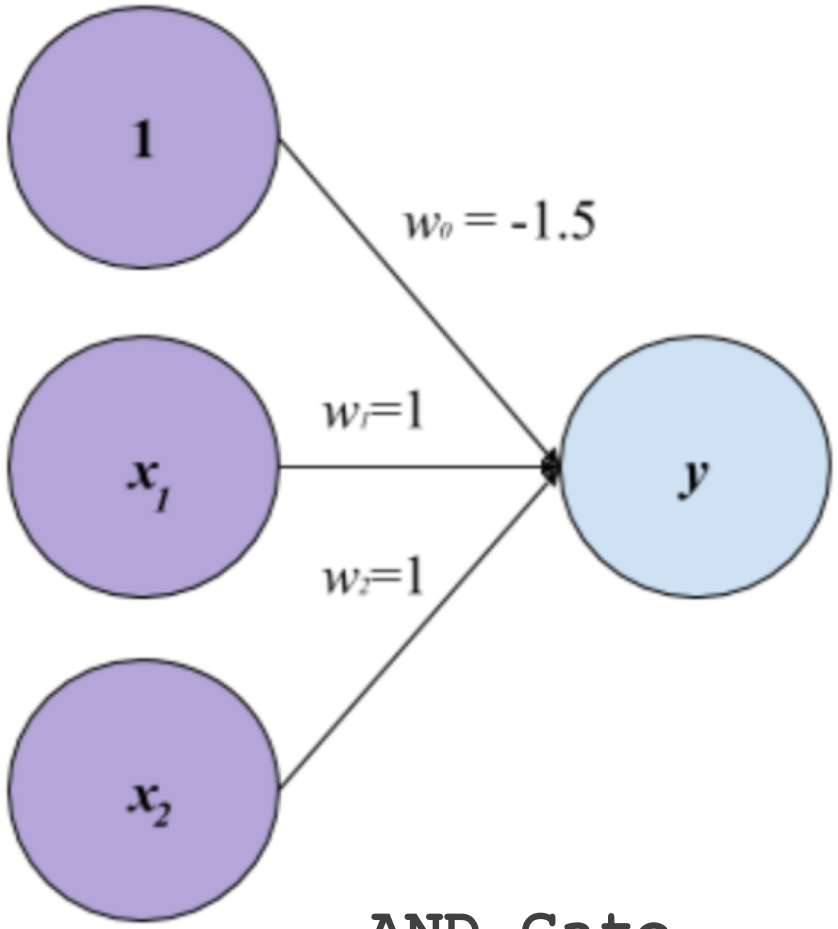
Perceptron



OR Gate

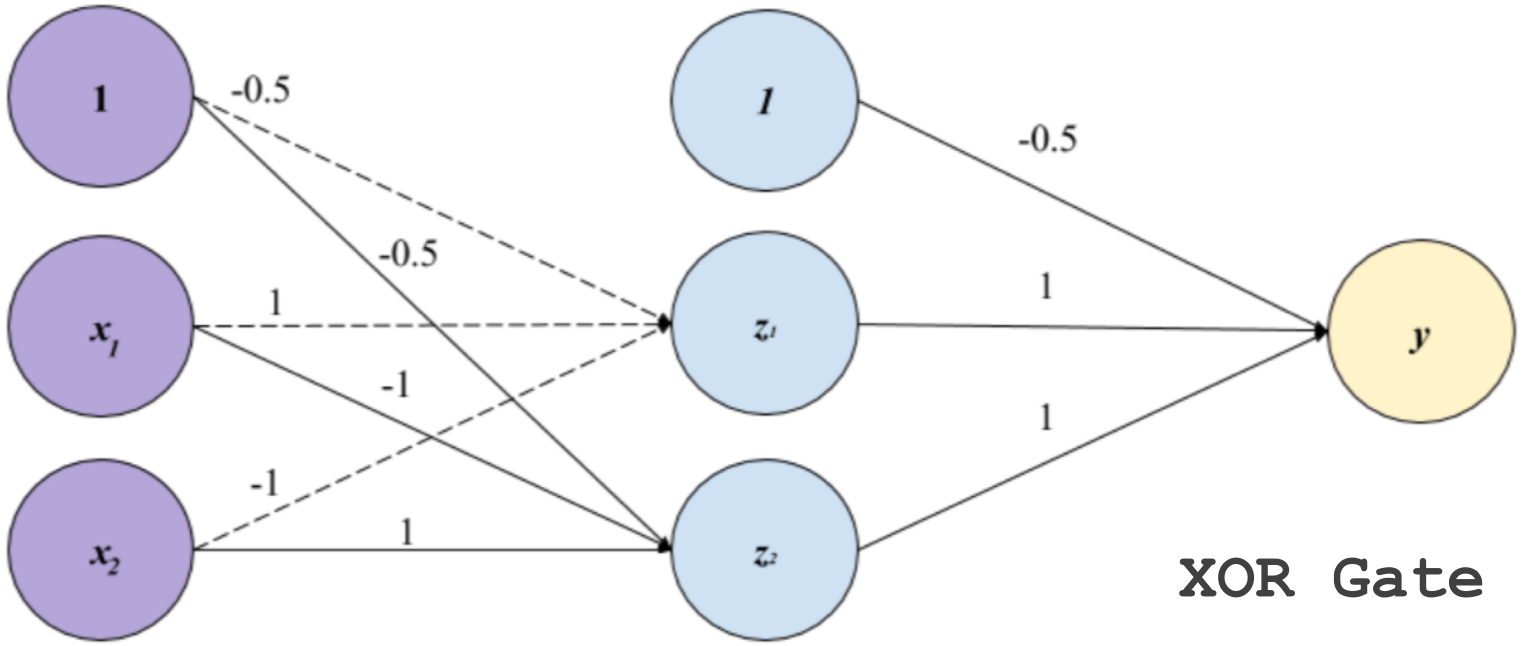
x_1	x_2	$sum = w_0*1 + w_1*x_1 + w_2*x_2$	$y = 1$ if $sum > 0$ $y = 0$ if $sum <= 0$
0	0	-0.5*1 + 1*0 + 1*0 = -0.5	0
0	1	-0.5*1 + 1*0 + 1*1 = 0.5	1
1	0	-0.5*1 + 1*1 + 1*0 = 0.5	1
1	1	-0.5*1 + 1*1 + 1*1 = 1.5	1

Perceptron



AND Gate

x_1	x_2	$sum = w_0*1 + w_1*x_1 + w_2*x_2$	$y = 1$ if $sum > 0$ $y = 0$ if $sum <= 0$
0	0	-1.5*1 + 1*0 + 1*0 = -1.5	0
0	1	-1.5*1 + 1*0 + 1*1 = -0.5	0
1	0	-1.5*1 + 1*1 + 1*0 = -0.5	0
1	1	-1.5*1 + 1*1 + 1*1 = 0.5	1



XOR Gate

x_1	x_2	$sum_1 = w_0*1 + w_1*x_1 + w_2*x_2$	z_1	$sum_2 = w_0*1 + w_1*x_1 + w_2*x_2$	z_2
0	0	-0.5*1 + 1*0 + -1*0 = -0.5	0	-0.5*1 + -1*0 + 1*0 = -0.5	0
0	1	-0.5*1 + 1*0 + -1*1 = -1.5	0	-0.5*1 + -1*0 + 1*1 = 0.5	1
1	0	-0.5*1 + 1*1 + -1*0 = 0.5	1	-0.5*1 + -1*1 + 1*0 = -1.5	0
1	1	-0.5*1 + 1*1 + -1*1 = -0.5	0	-0.5*1 + -1*1 + 1*1 = -0.5	0

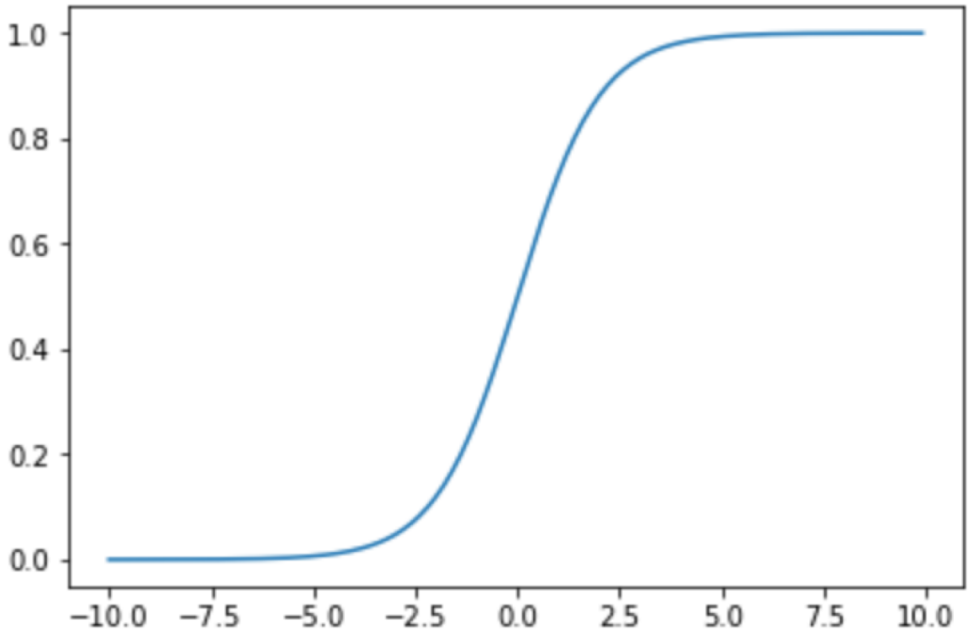
z_1	z_2	$sum_1 = w_0*1 + w_1*x_1 + w_2*x_2$	y
0	0	-0.5*1 + 1*0 + 1*0 = -0.5	0
0	1	-0.5*1 + 1*0 + 1*1 = 0.5	1
1	0	-0.5*1 + 1*1 + 1*0 = 0.5	1
0	0	-0.5*1 + 1*0 + 1*0 = -0.5	0

Figure 2: Emulate an XOR Gate with Feed-forward Network

Multi Layered Perceptron

Activation Function (Sigmoid)

```
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  def sigmoid(x):
7      return 1/(1+np.exp(-x))
8
9  # Generate points from -10 to +10,
10 # in steps of 0.1
11 x = np.arange(-10, 10, 0.1)
12 y = sigmoid(x)
13
14 # Plot the graph.
15 plt.plot(x, y)
16 plt.show()
```



Loss Function (aka. Criterion)

For regression problems, the simplest loss function is to simply take the difference between the predictions and the truth value, i.e. the L1 Loss / Mean Absolute Error (MAE)

Loss Function (aka. Criterion)

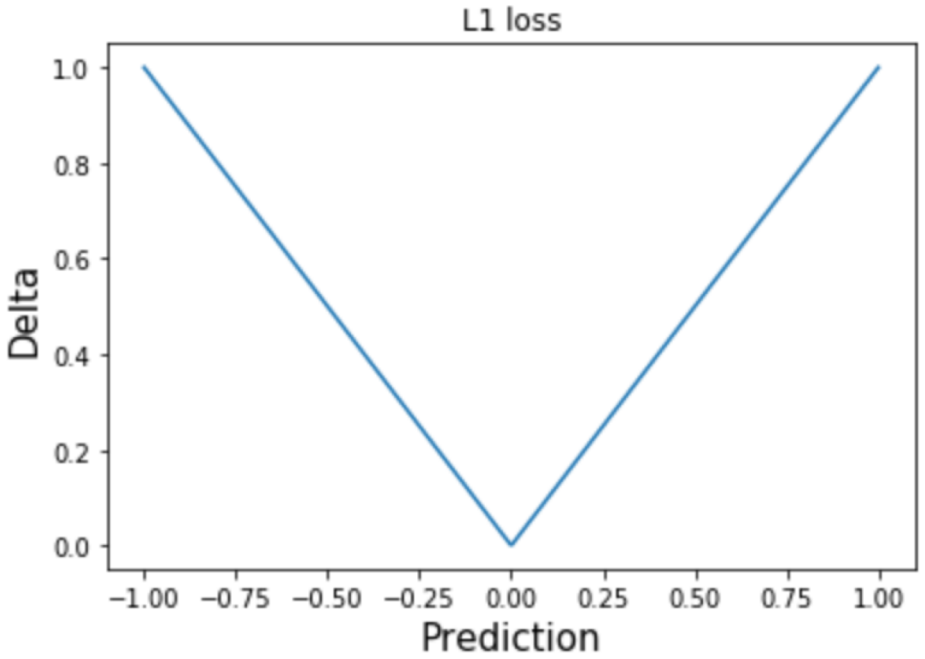
```
38  import numpy as np
39
40  # Create 500 points betwen -1 and 1.
41  predictions = np.linspace(-1, 1., 500)
42  # Set truth to be the constant 0.
43  truth = np.zeros(500)
44  # Calcuate the absolute differences
45  delta = np.abs(truth - predictions)
46
47  # Plotting magic
48  plt.plot(predictions, delta, 'b-', label='L1 loss' )
49  plt.title('L1 loss')
50  plt.xlabel('Prediction', fontsize=15)
51  plt.ylabel('Delta', fontsize=15)
```

For regression problems, the simplest loss function is to take the diff between predictions and the truth value, i.e. the L1 Loss / Mean Absolute Error (MAE)

Loss Function (aka. Criterion)

```
38  import numpy as np
39
40  # Create 500 points betwen -1 and 1.
41  predictions = np.linspace(-1, 1., 500)
42  # Set truth to be the constant 0.
43  truth = np.zeros(500)
44  # Calcuate the absolute differences
45  delta = np.abs(truth - predictions)
46
47  # Plotting magic
48  plt.plot(predictions, delta, 'b-', label='L1 loss' )
49  plt.title('L1 loss')
50  plt.xlabel('Prediction', fontsize=15)
51  plt.ylabel('Delta', fontsize=15)
```

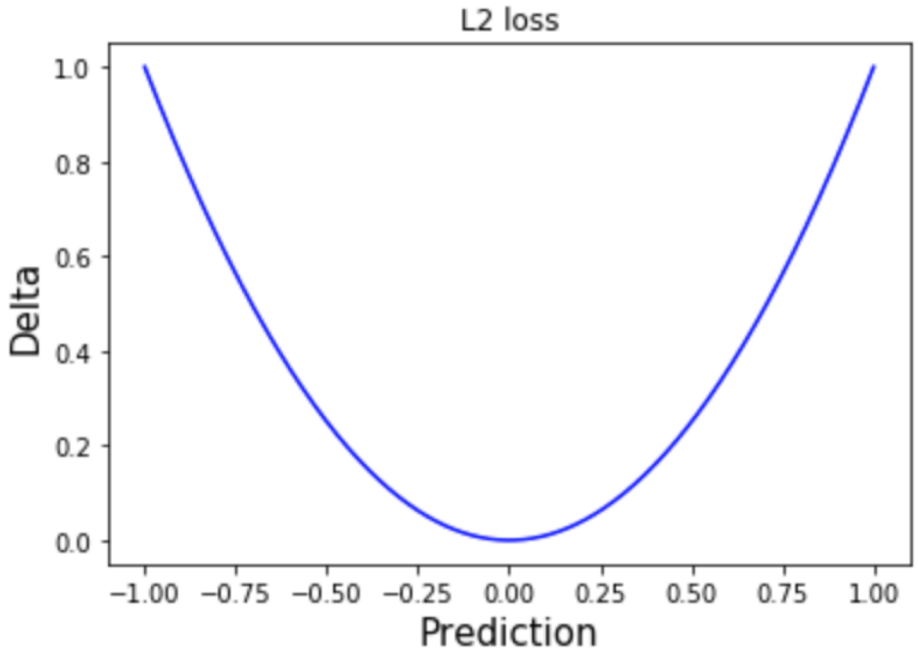
$$L_{mae} = \frac{1}{n} \sum_{i=1}^{n} |y - \hat{y}|$$



Loss Function (aka. Criterion)

```
38  import numpy as np
39
40  # Create 500 points betwen -1 and 1.
41  predictions = np.linspace(-1, 1., 500)
42  # Set truth to be the constant 0.
43  truth = np.zeros(500)
44  # Calcuate the absolute differences
45  delta = np.abs((truth - predictions)**2)
46
47  # Plotting magic
48  plt.plot(predictions, delta, 'b-', label='L2 loss' )
49  plt.title('L2 loss')
50  plt.xlabel('Prediction', fontsize=15)
51  plt.ylabel('Delta', fontsize=15)
```

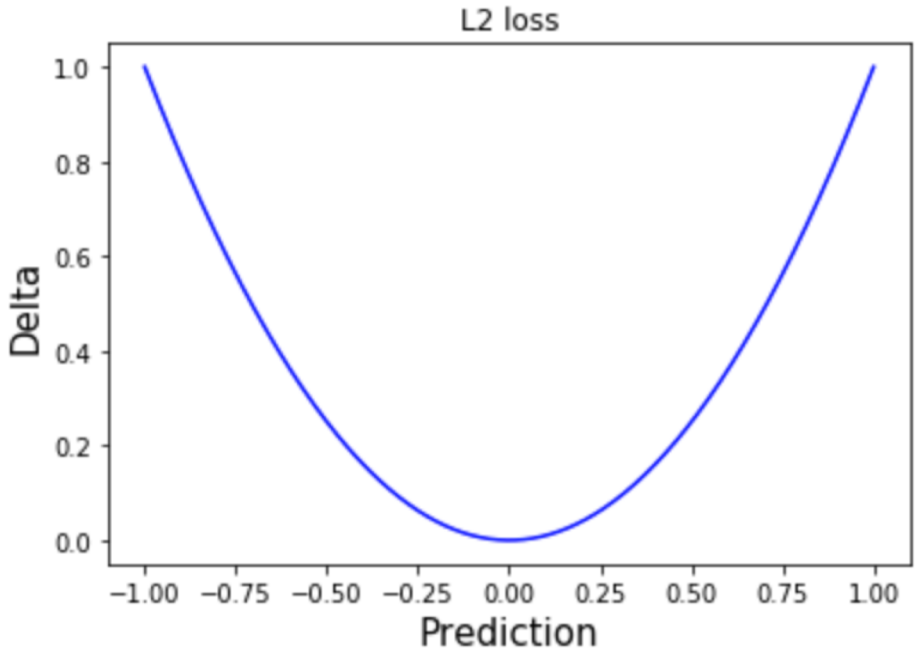
$$L_{mse} = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2$$



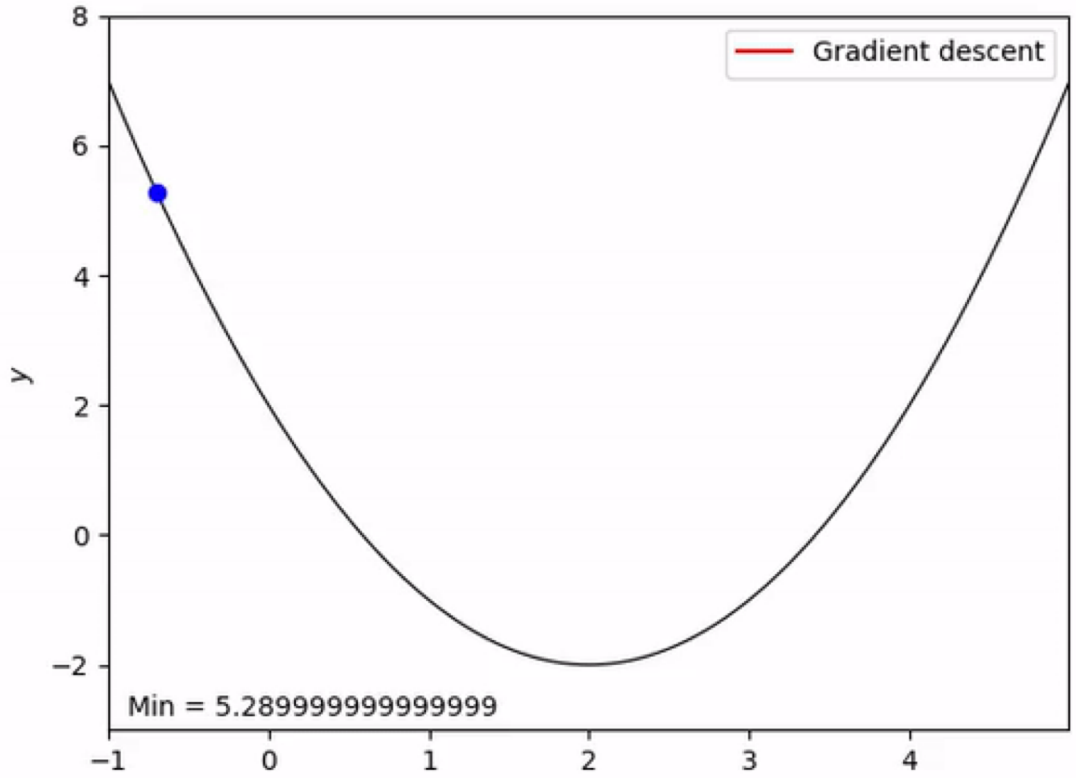
Loss Function (aka. Criterion)

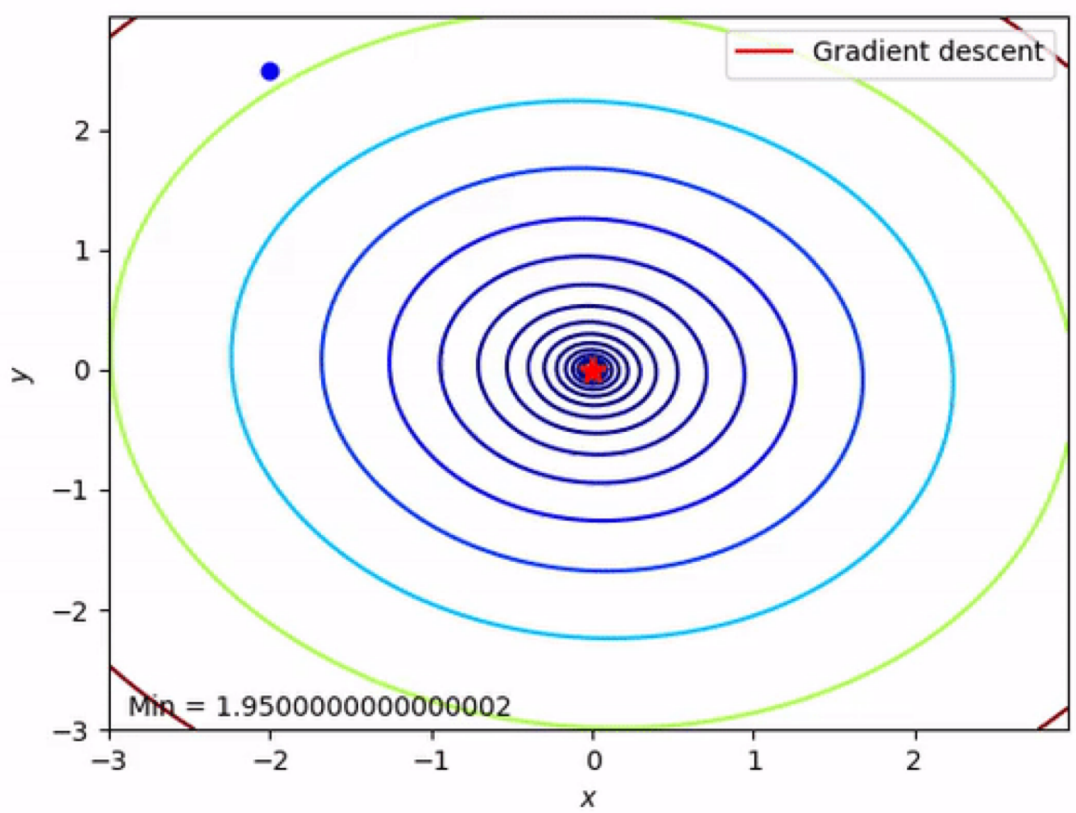
```
38  import numpy as np
39
40  # Create 500 points betwen -1 and 1.
41  predictions = np.linspace(-1, 1., 500)
42  # Set truth to be the constant 0.
43  truth = np.zeros(500)
44  # Calcuate the absolute differences
45  delta = np.abs((truth - predictions)**2)
46
47  # Plotting magic
48  plt.plot(predictions, delta, 'b-', label='L2 loss' )
49  plt.title('L2 loss')
50  plt.xlabel('Prediction', fontsize=15)
51  plt.ylabel('Delta', fontsize=15)
```

$$L_{mse} = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2$$



Optimization (Gradient Descent)





(Images from https://jed-ai.github.io/py1_gd_animation/)
It has some cool code to generate the GD pictures

Training Routine

Typically, process performs the following 4 steps iteratively.

Initialization

1. Initialize weights vector

Forward Propagation

2a. Multiply the weights vector with the inputs, sum the products.
2b. Put the sum through the activation function, e.g. sigmoid

Training Routine

Back Propagation

3a. Compute the errors, i.e. difference between expected output and predictions
3b. Multiply the error with the derivatives to get the delta
3c. Multiply the delta vector with the inputs, sum the product

Optimizer takes a step

4. Multiply the learning rate with the output of step 3c

Repeat 1-4 until *desired*

Summary

Summary

- **Classic NLP == Lots of feature engineering**
  - Frequency, TF-IDF, PPMI
- **Deep 'Magic' NLP**
  - Transfer Learning moment for NLP
  - NMT achieving human parity & Unsupervised MT
  - Robustness and Interpretability
- **Deep Learning Basics**
  - Perceptron and Multi-layered Perceptron
  - Activation function, Loss function, Gradient Descent
  - Training routine

Overview

Lecture

- Classic NLP (40 mins)
- Deep Magic NLP (20 mins)
- Deep Learning Basics (30 mins)

Hands-on

- Environment Setup (15 mins)
- Deep Learning From Scratch (60 mins)

Course Logistics

Open Anaconda Navigator.

Go to the PyTorch installation page, copy the command as per configuration: https://pytorch.org/get-started/locally/

Fire up the terminal in Anaconda Navigator.

Start a Jupyter Notebook.

Download https://goo.gl/fyZv3m

Import the .ipynb to the Jupyter Notebook

Fin

www.iss.nus.edu.sg

facebook.com/ISS.NUS

twitter.com/ISSNUS

@iss.nus

linkedin.com/company/iss.nus

youtube.com/user/TheISSNUS/